PATENT ABSTRACTS OF JAPAN

(11)Publication number : **2001-109788**
(43)Date of publication of application : **20.04.2001**

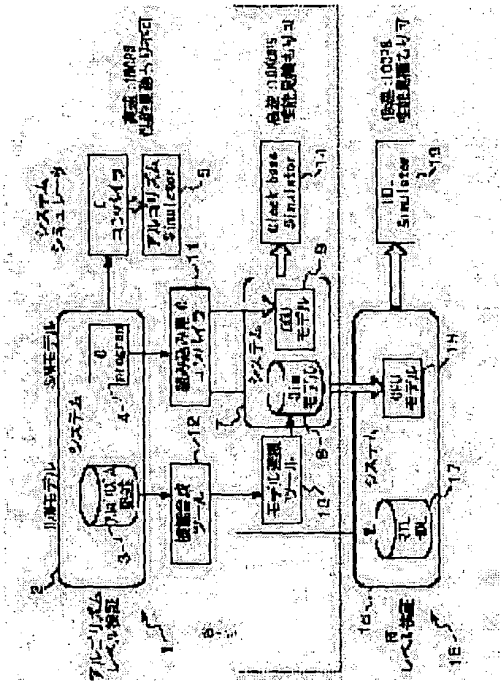
(51)Int.Cl. G06F 17/50

(21)Application number : **11-290276**
(22)Date of filing : **12.10.1999**

(71)Applicant : **NEC CORP**
(72)Inventor : **IKEGAMI HIROYUKI**

(54) SIMULATION MODEL, ITS GENERATING METHOD, SIMULATION METHOD AND ITS RECORDING MEDIUM

(57)Abstract:
PROBLEM TO BE SOLVED: To execute the simulation of a middle level between an algorithm description and an RT level description and to provide its language.
SOLUTION: The algorithm description 3 is degraded to a clock level description 8 under the restriction of resources. The plurality of functions in the algorithm description 3 are disassembled into partial functions which are operated in a unit clock and the partial functions are assembled to restore the plurality of functions. The plurality of functions are expressed in a language where a register is adopted as a variable number as a clock level simulator 8 being the clock level description. The language with the register as the variable number is optimum for the language of programming to be operated by clock unit, is in lower order than the algorithm level and is the indiscovable language in higher order than the RT level.



LEGAL STATUS

[Date of request for examination]	04.09.2000
[Date of sending the examiner's decision of rejection]	05.08.2002
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	
[Date of registration]	
[Number of appeal against examiner's decision of rejection]	2002-16958

[Date of requesting appeal against examiner's decision of rejection] 04.09.2002

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号
特開2001－109788
（P2001－109788A）
(43)公開日　平成13年４月20日(2001.4.20)

(51)Int.Cl.⁷	識別記号	Ｆ Ｉ	テーマコード（参考）
Ｇ０６Ｆ　17/50		Ｇ０６Ｆ　15/60	６６４Ｋ　５Ｂ０４６
			６５４Ａ
			６７２Ｚ

審査請求　有　　請求項の数10　ＯＬ　（全 18 頁）

(21)出願番号　特願平11－290276

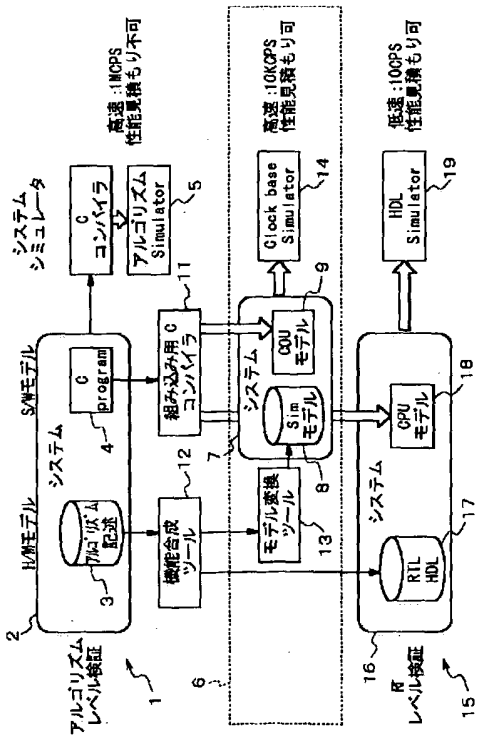
(22)出願日　平成11年10月12日(1999.10.12)

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(72)発明者　池上　裕之
東京都港区芝五丁目７番１号　日本電気株式会社内
(74)代理人　100102864
弁理士　工藤　実　（外１名）

Ｆターム(参考)　5B046 AA08 BA02 BA03 DA04 GA01
JA04 JA05

(54)【発明の名称】　シミュレーションモデル、その生成方法、シミュレーション方法及びその記録媒体

(57)【要約】
【課題】アルゴリズム記述とＲＴレベル記述の中間のレベルのシミュレーションを可能にする。その言語の提供。
【解決手段】資源の制約下、アルゴリズム記述３をクロックレベル記述８に低位化する。アルゴリズム記述３の複数機能を単位クロックの中で動作が可能である部分機能に分解し、その複数機能を回復するためにその部分機能を組み立てる。複数機能は、クロックレベル記述であるクロックレベルシミュレータ８として、レジスタを変数とする言語により表現されている。レジスタを変数とする言語は、クロック単位で動作するプログラミングの言語として最適であり、この言語は、アルゴリズムレベルより下位であり、ＲＴレベルよりも上位の未発見言語であった。



1

【特許請求の範囲】

【請求項１】資源の制約の下で、回路のアルゴリズム記述の複数機能を単位クロックの中で動作が可能である部分機能に分解することと、

前記複数機能を形成するために前記部分機能を組合わせることにより前記アルゴリズム記述からクロックレベル記述を生成することとを含むことを特徴とするシミュレーションモデルの生成方法。

【請求項２】前記クロックレベル記述は、

前記資源のうちのレジスタを異なる複数機能間で兼用させる記述を備える請求項１記載のシミュレーションモデルの生成方法。

【請求項３】請求項１又は２に記載されるシミュレーションモデルの生成方法により生成されたクロックレベル記述と、

前記クロックレベル記述に対応する前記アルゴリズム記述とを対応させ単位クロックに相当する状態毎の資源使用対応表とを含むことを特徴とするシミュレーションモデル。

【請求項４】請求項３に記載されるシミュレーションモデルを用いて前記単位クロックで前記部分機能を動作させることにより前記複数機能をシミュレートすることを含むシミュレーション方法。

【請求項５】前記シミュレートすることの結果に基づいて、状態と前記アルゴリズム記述行とを対応づけ、前記アルゴリズム記述行毎の資源使用状況を表示することを特徴とする請求項４記載のシミュレーション方法。

【請求項６】前記シミュレーションモデルに含まれるクロックレベル記述と対応表とを用いて、前記アルゴリズム記述行と当該シミュレーションにより得られる変数値の対応を表示することを特徴とする請求項３記載のシミュレーション方法。

【請求項７】前記クロックレベル記述とクロックレベル記述のうちの前記レジスタに関するレジスタ記述との対応表を作成することと、

前記クロックレベル記述の前記クロック単位の状態遷移が起これば前記状態遷移が起こった前記アルゴリズム記述部分を表示し、又は、前記クロックレベル記述の前記クロック単位の状態遷移が起これば前記状態遷移が起こった前記レジスタ記述部分の前記レジスタの変数値を表示することの少なくとも一方を実行することとを含むことを特徴とする請求項２記載のシミュレーションモデルの生成方法。

【請求項８】請求項３に記載するシミュレーションモデル記述を記録するシミュレーション用記録媒体。

【請求項９】アルゴリズム記述と、

前記アルゴリズム記述よりも抽象度が低いＲＴレベル記述と、

前記アルゴリズム記述よりも抽象度が低く、且つ、前記ＲＴレベル記述よりも抽象度が高いクロックレベル記述を含み、

前記クロックレベル記述は前記アルゴリズム記述からクロック単位で生成する記述であり、

前記アルゴリズム記述と、前記クロックレベル記述と、前記ＲＴレベル記述とは、前記アルゴリズム記述に含まれて記述されるレジスタの変数値を共有するシミュレーションモデル。

【請求項１０】前記クロックレベル記述は演算を含み、時間軸上で進行するクロックの中で実行する前記演算の変数がレジスタである請求項３記載のシミュレーションモデル。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、シミュレーション方法、シミュレーションモデル、これらに対応する記録媒体、それらの記述の生成方法に関し、特に、アルゴリズム記述とＲＴレベル記述との間でシミュレーションを適宜に切り換えて高速度で所望の段階でシミュレーションモデルを生成してアルゴリズムを高速に又は詳細にシミュレートするシミュレーション方法、シミュレーションモデル、それらに対応する記録媒体、それらの記述の生成方法に関する。

【０００２】

【従来の技術】大規模回路は、自動設計装置により設計される。自動設計装置の設計フローは、Ｃ言語のような汎用プログラム言語、専用の動作レベル記述言語により所望の動作フローが記述された最高位レベル記述をレジスタ、加算器のようなハードウエア資源を用いてよりハードウエア化されるＲＴレベル記述のような低位レベル記述に書き下すステップを有している。このようにレベルが異なる抽象度の複数のシミュレーションモデルのシミュレーションは、図１３（ａ），（ｂ）に示されるように、抽象度が高い動作レベルシミュレーションモデル１０１では、初めから終わりまでこれだけがシミュレートされ、そのシミュレーションの途中で、ＲＴレベルシミュレーションモデル１０２のシミュレーションは行われえず、また、抽象度が低いＲＴレベルシミュレーションモデル１０２では、初めから終わりまでこれだけがシミュレートされ、そのシミュレーションの途中で、動作レベルシミュレーションモデル１０１のシミュレーションは行われなかった。

【０００３】このように、動作レベルシミュレーションモデル１０１とＲＴレベルシミュレーションモデル１０２は、互いに独立していて別個のものとして扱われ、シミュレーションの途中で一方のシステムシミュレーションから他方のシステムシミュレーションに切り換えることは不可能であった。一般的には、速度が重要な局面では高速な上位シミュレーションモデルが用いられ、精度が重要な局面ではＲＴレベル、精度が更に細かく要求される局面ではレベルが更に低い他の低位シミュレーショ

ンモデルがそれぞれに用いられていた。

【０００４】しかし、機能検証のシミュレーションを行う場合、ある時刻ｔ１から他のある時刻ｔ２の間を特に詳しく調べたいという要望がある。そのような場合にも時刻ｔ０から時刻ｔ２までの検証のために高精度のシミュレータを使用すると、機能検査に多大な時間がかかってしまう。

【０００５】このような問題を解決する技術として、異なる抽象度の複数・シミュレーションモデルの間で切り換えを行うようにしたミックス・シミュレーション技術が、特開平５－６１９３４号、論文「並列論理シミュレータＷＩＺＤＯＭ」（情報処理学会第５７回全国大会、１９９８）で知られている。ここで開示されている切り換えは、命令レベルシミュレーションモデルとハードウエアシミュレータ、ＲＴレベルシミュレーションモデルのような抽象度が異なるシミュレーションモデルとの間の切り換えである。このような切り換えが可能になっている根拠は、双方のモデルがレジスタ等で共通する構造を持っていることにある。また、ゲートレベルと電子回路レベルの間でのシミュレーション切り換えが行われる技術が、特開平７－１１０８２６号で知られている。この切り換えで双方の情報のトランスファーは、ゲート回路の端子と電子回路の端子が１対１に対応していることを利用することにより、回路端子のレベルのアナログ・ディジタル変換によって行っている。また、特開平１０－２６１００２号は、ある回路記述形式を用いてその回路の動きを詳しい定義モデルと粗い定義モデルとで表現し、そのモデルの間での切り換えを開示している。このような１対１対応の高低レベル間の切り換え、詳細度の異なるものの共通の記述形式を用いた同一レベル間の切り換えが知られている。

【０００６】しかし、動作記述レベルといわれる最高位のプログラム言語記述レベルと下位記述レベルであるＲＴレベルの間では、シミュレーション状態を保持する構造が１対１に対応しておらず、両レベル間でデータの受け渡しができないため、従来、これらのレベル間の切り換えシミュレーションが不可能であった。シミュレーションをより適正に高速化し且つ詳細化するために、このように１対１に対応する記述がなされていないレベル間のシミュレーションモデルの切換技術の確立が特に望まれた。

【０００７】本発明者は、そのような技術の確立のために、論理回路の処理フローを記述した動作レベル記述を機能合成ツールによりＲＴレベル記述に変換し、その動作レベル記述とＲＴレベル記述をそれぞれに対応するコンパイラに入力して動作レベルシミュレーションモデル構造とＲＴレベルシミュレーションモデル構造を生成し、動作レベルシミュレーションモデル構造とＲＴレベルシミュレーションモデル構造の対応手段を生成し、動作レベルシミュレーションモデル構造とＲＴレベルシミュレーションモデル構造に基づいて動作レベルシミュレーションとＲＴレベルシミュレーションを対応手段を介して切り換えて実行するように、両レベル間でレジスタの変数値を共有させるようにしたシミュレーション方法を提案している（参照：特願平１１－０５７０４０号）。

【０００８】このようなシミュレーション方法によるシミュレーションの実行過程で、本発明者は、動作モデルである最高位の抽象度を持つアルゴリズム記述から自動生成される従来のＲＴレベル記述は、ソースであるアルゴリズム記述から遠くかけ離れ、両記述の間に大きな抽象度の落差が存在していることに気づいた。両記述の間に潜み両記述のそれぞれの抽象度の中間にある抽象度の記述の発見が望まれ、その記述により、アルゴリズム記述より抽象度が低くＲＴレベルより抽象度が高い言語レベルでシミュレーションモデルを動作レベルより精細に且つＲＴレベルより高速にシミュレートすることが望まれる。

【０００９】

【発明が解決しようとする課題】本発明の課題は、アルゴリズムレベルより抽象度が高くＲＴレベルよりも抽象度が低いレベルの言語を用いてそれに基づくシミュレーションモデル、その記述とその生成の方法を提供することにある。本発明の他の課題は、アルゴリズムレベルより抽象度が高くＲＴレベルよりも抽象度が低い記述に基づいて、アルゴリズム記述のシミュレーションよりもより精細に、且つ、ＲＴレベル記述のシミュレーションよりもより高速にシミュレートすることができるシミュレーションモデル、その記述とその生成の方法を提供することにある。

【００１０】

【課題を解決するための手段】その課題を解決するための手段が、下記のように表現される。その表現中に現れる技術的事項には、括弧（）つきで、番号、記号等が添記されている。その番号、記号等は、本発明の実施の複数・形態又は複数の実施例のうちの少なくとも１つの実施の形態又は複数の実施例を構成する技術的事項，特に、その実施の形態又は実施例に対応する図面に表現されている技術的事項に付せられている参照番号、参照記号等に一致している。このような参照番号、参照記号は、請求項記載の技術的事項と実施の形態又は実施例の技術的事項との対応・橋渡しを明確にしている。このような対応・橋渡しは、請求項記載の技術的事項が実施の形態又は実施例の技術的事項に限定されて解釈されることを意味しない。

【００１１】本発明によるシミュレーションモデルの記述生成方法は、資源の制約の下で、アルゴリズム記述（３）をクロックレベル記述（８）に低位化することを含み、その低位化することは、アルゴリズム記述（３）の複数機能を単位クロックの中で動作が可能である部分

機能に分解することと、その複数機能を回復するためにその部分機能を組み立てることとを備える。そのような複数機能は、クロックレベル記述であるクロックレベルシミュレータ（８）として、後述されるレジスタを変数とする言語により表現されている。レジスタを変数とする言語は、本発明者により発見された低位プログラミング記述言語である。

【００１２】クロックレベル記述は、資源のうちの複数レジスタの内のある１つのレジスタを異なる複数単位クロックのうちで動作させる記述を備える。このようなクロック間でレジスタの共有化が行われ、レジスタが変数として言語化される。

【００１３】本発明によるシミュレーションモデルは、更に、既述のこのようなシミュレーションモデルの記述生成方法により記述されたクロックレベル記述（８）と、そのクロックレベル記述に対応するアルゴリズム記述（３）とが対応する対応表（２２，２３）とを含む。

【００１４】本発明によるシミュレーション方法は、このようなシミュレーションモデルを用いて単位クロックで部分機能を動作させることにより複数機能をシミュレートすることを含み、更に、そのようにシミュレートすることの結果に基づいて、アルゴリズム記述（３）をデバッグすることを含む。更には、シミュレーションモデルに含まれるクロックレベル記述と対応表とを用いて、アルゴリズム記述の行数とシミュレーションにより得られる変数値の対応を表示することを含む。

【００１５】本発明によるシミュレーションモデルの生成方法は、資源の制約の下で、アルゴリズム記述（３）をクロックレベル記述（８）に低位化することを含み、その低位化することは、アルゴリズム記述（３）の複数機能を単位クロックの中で動作が可能である部分機能に分解することと、複数機能を回復するためにその部分機能を組み立てることとを備え、クロックレベル記述（８）は、その資源のうちの複数レジスタの内のある１つのレジスタを異なるクロックのうちでそれぞれに動作させる記述を備え、更に、クロックレベル記述（８）とクロックレベル記述（８）のうちのレジスタ（Ｒｅｇ１～５）に関するレジスタ記述との対応表（２２，２３）を作成することと、クロックレベル記述（８）のクロック単位の状態遷移が起こればその状態遷移が起こったアルゴリズム記述部分を表示することと、クロックレベル記述（８）のクロック単位の状態遷移が起こればその状態遷移が起こったレジスタ記述部分のレジスタ（Ｒｅｇ１～５）の変数値（Ｒ）を表示することとを含む。このような対応表の作成によって、両レベル間の移行が可能である。このようなシミュレーションモデルの生成方法に含まれる全記述がコンピュータにより読み取り可能であることは当然であり、それは記録媒体化されて利用に供される。

【００１６】本発明によるシミュレーションモデルは、アルゴリズム記述（３）と、アルゴリズム記述（３）よりも抽象度が低いＲＴレベル記述（１７）と、アルゴリズム記述（３）よりも抽象度が低く、且つ、ＲＴレベル記述（１７）よりも抽象度が高いクロックレベル記述（８）を含み、クロックレベル記述（８）はアルゴリズム記述（３）からクロック単位で自動生成される記述であり、アルゴリズム記述（３）と、クロックレベル記述（８）と、ＲＴレベル記述（１７）とは、クロックレベル記述（８）に含まれて記述されるレジスタ（Ｒｅｇ１～５）の変数値（Ｒ）を共有する。このような共有により、３つのレベル間で移行が可能である。

【００１７】更に、シミュレーションモデルは、アルゴリズム記述（３）と、アルゴリズム記述（３）よりも抽象度が低いクロックレベル記述（８）を含み、クロックレベル記述（８）は、時間軸に直交する単位クロックごとの断面上で記述されるレジスタの演算記述である。ここで、断面とは、１単位クロック内で同時的に動作が進行する一連の記述の連鎖である。アルゴリズム記述よりも抽象度が低く、且つ、時間軸に直交する単位クロックごとの断面上でレジスタに関して記述される言語は、コンピュータにより読み取り可能に表現されて記録媒体化され得る。このような媒体をコピーすることにより、同時並行的に多様なシミュレーションが可能である。アルゴリズム記述よりも抽象度が低く、且つ、時間軸に直交する単位クロックごとの断面上でレジスタに関して記述されるシミュレーションモデル用記述言語は、Ｒｅｇ１＋Ｒｅｇ２のように複数レジスタが用語として多変数化されている。アルゴリズム記述（３）は数の演算を含んでいる場合には、その演算を時間軸上で進行するクロックの単位の中で実行する関数の変数がレジスタになって表現されている。

【００１８】このように、中間レベルの抽象度の言語には、レジスタの動作がクロック単位で現れる変数が潜んでいる。

【００１９】

【発明の実施の形態】図１に一致対応して、本発明によるシミュレーションモデルの実施の形態は、アルゴリズム検証部とＲＴレベル検証部との間で、新たにクロックレベル検証部が設けられていることを特徴としている。そのアルゴリズム検証部１は、図１に示されるように、アルゴリズムシステム２から形成されている。アルゴリズムシステム２は、最も抽象度が高いＨ／Ｗモデルであるアルゴリズム記述３と、Ｃ言語で記述されＳ／Ｗモデルであるｃ－プログラム４とを備えている。アルゴリズムシステム２は、Ｃ－コンパイラによりアルゴリズムシミュレータ５に変換される。アルゴリズムシステム２は、アルゴリズムシミュレータ５によりシミュレートされる。

【００２０】そのクロックレベル検証部６は、クロックレベルシステム７から形成されている。クロックレベル

システム７は、クロックレベル記述であるクロックレベルシミュレーションモデル８と、クロックレベルＣＰＵモデル９とを備えている。クロックレベルシステム７は、組込用Ｃコンパイラ１１により自動生成されて記述変換され、その記述変換には、慣用の機能合成ツール１２が持つツールであるレジスタが用語（単語又は一次変数）として用いられる。アルゴリズム記述３は、機能合成ツール１２で記述されるツールを持つモデル変換ツール１３によりクロックレベルシミュレーションモデル８に記述変換される。クロックレベルＣＰＵモデル９は、組込用Ｃコンパイラ１１によりＣ－プログラム４から自動生成される。クロックベースシミュレータ１４は、クロックレベルシミュレーションモデル８とクロックレベルＣＰＵモデル９とから形成されている。

【００２１】そのＲＴレベル検証部１５は、ＲＴレベルシステム１６から形成されている。ＲＴレベルシステム１６は、ＲＴＬ－ＨＤＬ１７と、ＲＴレベルＣＰＵモデル１８とを備えている。ＲＴＬ－ＨＤＬ１７は、機能合成ツール１２によりアルゴリズム記述３から自動生成される。ＨＤＬシミュレータ１９は、ＲＴＬ－ＨＤＬ１７とＲＴレベルＣＰＵモデル１８とから形成されている。

【００２２】図２は、クロックレベルシミュレーションモデル８を自動生成するモデル作成のツール構成とその作成フローを示している。機能合成ツール１２は、定数、変数の最適化機能と、スケジューリング機能と、アロケーション機能と、レジスタシェアリング機能と、ＨＤＬ生成機能とを有している。レジスタシェアリング機能は、下記３つの作成を実行する機能を有している：

（１）資源制約下で用いられる複数レジスタ資源の状態遷移の繰返しをデータパス制御により制御するＦＳＭ／ＤａｔａＰａｔｈモデル２１の作成

（２）変数とレジスタと状態位置との対応である変数／レジスタ／状態位置対応表２２の作成

（３）アルゴリズム記述のソース行とその状態位置との対応であるソース行／状態位置対応表２３の作成

【００２３】モデルＩ／Ｆ情報２４は、機能合成ツール１２とモデル変換ツール１３に入力される。モデル変換ツール１３は、モデルＩ／Ｆ情報２４とＦＳＭ／ＤａｔａＰａｔｈモデル２１と変数／レジスタ／状態位置対応表２２とに基づいて動作し、アルゴリズム記述３をクロックレベルシミュレーションモデル８に記述変換して、クロックレベルシミュレーションモデル８を自動生成する。モデル変換ツール１３には、その自動生成のために、モデル機能ライブラリのデバッグ機能ライブラリ２５とモデルＩ／Ｆライブラリ２６からそれらのライブラリ情報が入力される。

【００２４】図３は、クロックレベルシミュレーションモデル８の記述構造を示している。クロックレベルシミュレーションモデル８は、Ｂｕｓシミュレーションモデル３１を備えている。Ｂｕｓシミュレーションモデル３１は、モジュールＩＯ部３２とデータパス記述部３３とから形成されている。モジュールＩＯ部３２には、複数ＩＯレジスタ３４と複数ＩＯメモリー３５とが記述され、モデルＩ／Ｆ情報２４から作成され、当該シミュレーションモデルの入出力端子を通して組込ソフトウエアから読み書きされるＩＯレジスタ構造と、ＩＯメモリー構造を有している。クロックレベルシミュレーションモデル８は、モジュールＩＯ部を介して、他のモジュールと信号伝達を行う。データパス記述部３３は、後述されるように、複数ＩＯレジスタと複数演算子との関係を記述するデータパスと、その演算のクロック単位の動作を制御する制御構造とを有している。

【００２５】データパス記述部３３に、シミュレーションコントローラ３６が接続している。シミュレーションコントローラ３６は、シミュレータ本体からクロック入力３７を受けてデータパス記述部３３のクロックを１つずつ進める演算動作遷移３８の循環歩進制御（ＦＳＭ制御）を行う。シミュレーションコントローラ３６から出力されるＦＳＭ制御信号３９は、データパス記述部３３に入力される。

【００２６】データパス記述部３３は、ＧＵＩコントローラ４１に接続している。データパス記述部３３は、レジスタ値Ｒと演算遷移位置である状態位置値Ｓとを出力する。レジスタ値Ｒと状態位置値Ｓとは、ＧＵＩコントローラ４１に入力される。リセット信号がシミュレーションコントローラ３６に入力されると、データパス記述部３３のレジスタ値Ｒと状態位置値Ｓとは初期化される。

【００２７】ＧＵＩコントローラ４１は、現在の遷移状態位置のレジスタとアルゴリズムレベル記述による変数との対応を図２に示される変数／レジスタ／状態位置対応表２２から得ることができる。ＧＵＩコントローラ４１は、更に、現在の遷移状態位置のアルゴリズムレベル記述によるソース行をソース行／状態位置対応表２３から得ることができる。

【００２８】ＧＵＩコントローラ４１は、アルゴリズムレベル記述変数値表示ウインドウ４２と、アルゴリズムレベル記述ソース実行行表示ウインドウ４３に接続している。アルゴリズムレベル記述変数値表示ウインドウ４２には、レジスタのレジスタ値Ｒが、その遷移状態位置で、アルゴリズムレベル記述変数値として変数／レジスタ／状態位置対応表２２に基づく対応関係が与えられて表示される。複数レジスタが部分的に異なる遷移状態間で共有されることにより一時的にレジスタとの対応が消えてしまうアルゴリズムレベル記述変数の値は、ＧＵＩコントローラ４１に保持されることにより、アルゴリズムレベル記述変数値表示ウインドウ４２には、常に最新の遷移状態の最新の変数値が表示される。

【００２９】他方で、アルゴリズムレベル記述ソース実行行表示ウインドウ４３には、現在の状態遷移位置に対

応するアルゴリズムレベル記述のソース行位置値Ｓが、現在の遷移状態位置で、ソース行／状態位置対応表２３に基づく対応関係が与えられて、ハイライト表示される。更に、ＧＵＩコントローラ４１は、アルゴリズム記述変数、及び、レジスタ、クロック単位の変化の回数、状態遷移の各状態に遷移した回数、遷移元、遷移先が、シミュレーション中に計測されることにより、変数、レジスタ変化、状態変化の網羅率が測定される。このような網羅率は、テストパタンによりどれ程網羅的な検証が行われ得たかの指標になる。

【００３０】アルゴリズム検証：システム内に含まれるハードウエアモデルとソフトウエアモデルは、共にプログラミング言語で記述されている。そのハードウエアモデルは、ハードウエアのアルゴリズムを表現し、そのソフトウエアモデルは組み込みソフロウエアを表現している。これらのプログラミング言語記述をＣコンパイラにかけることにより、アルゴリズムレベルのシミュレータが作成される。ハードウエアに関する情報がないアルゴリズムレベルで検証可能な項目は、下記（１），（２）に示されるように、純粋に論理的な動作のみであるが、そのシミュレーションは非常に高速である。

（１）各モジュールの論理的な動作の検証

（２）システムの論理的な動作の検証

【００３１】クロックレベル検証：ハードウエアモデルのアルゴリズム記述を機能合成ツール、モデル変換ツールで処理することにより、クロックレベルのシミュレーションモデルが作成される。ソフトウエアモデルは、ＣＰＵのシミュレーションモデル上に組込ソフトウエアとして読み込まれ、実際のＣＰＵと同等な動作タイミングを実現する。これらのハードウエア、ソフトウエアモデルは、クロックレベルで実際のＬＳＩシステムと同等のタイミング精度を持つため、クロックレベルシミュレータ上でシミュレーションを行うことにより、下記のような項目の見積もり、検証が可能である。

【００３２】（１）各モジュールのクロックレベルの動作タイミング検証

（２）各モジュールのインターフェースの概略検証（ＩＯレジスタ、ＩＯメモリ、ＩＯ端子の構成、名称、ビット幅等）

（３）各モジュール、バスの動作クロックの周波数見積もり

（４）キャッシュアクセスの見積もり（キャッシュヒット率、アクセス率、ライトバック回数等）

（５）アクセスの見積もり（バスの占有率、バスのトランザクションごとのＡｄｒｓ、Ｄａｔａ、Ｍａｓｔｅｒ、Ｓｌａｖｅ、Ｒｅａｄ／Ｗｒｉｔｅ、Ｃｏｍｍａｎｄ、語数、占有時間等）

（６）Ｂｕｓ、Ａｒｂｉｔｅｒのアルゴリズムの検証

（７）メモリ／ＩＦのトラフィックの見積もり

（８）データ処理のＴｈｒｏｕｇｈ　Ｐｕｔの見積もり（モジュール毎、バス毎、システム全体）

（９）バッファ、スタックサイズの見積もり

（１０）画質、音質の見積もり

（１１）アドレスマップ、端子、ｂｉｔ幅等モジュール間接続Ｉ／Ｆの整合性の検証

（１２）浮動小数点を固定小数点に変換した場合の見積もり

（１３）組み込みソフトウエアの開発、デバッグ

（１４）消費電力の概略見積もり

【００３３】ＲＴレベル検証：ハードウエアモデルは機能合成ツールが作成するＲＴＬ－ＨＤＬモデルが用いられる。このモデルは、クロックレベルよりも詳細に非同期的動作を含むタイミングの精度を持つ。ソフトウエアモデルは、クロックレベルシミュレーションと同様にＣＰＵのシミュレーションモデル上に組み込みソフトウエアとして読み込まれる。これらのＲＴＬ－ＨＤＬモデルとＣＰＵモデルをＨＤＬシミュレータ上でシミュレートすることにより、以下のような項目の検証と見積もりを行うことができる。

（１）各モジュールのインターフェースの詳細検証（制御、アドレス、データ端子のタイミング動作）

（２）各モジュールの詳細タイミングの検証

（３）消費電力の詳細見積もり

（４）テスタ向けパタン作成

【００３４】

【実施例】図１０は、本発明によるシミュレーションモデル作成方法の実施の形態を示している。図１０に示すフロー図にそって、図４に示されるアルゴリズム記述から後述するＦＳＭ／ＤａｔａＰａｔｈモデル記述及び図８に示される変数／レジスタ／状態位置対応表、図９に示されるソース行／状態位置対応表を作成し、クロックレベルシミュレーションモデルを作成する方法を以下に述べる。

【００３５】図４は、アルゴリズム記述３（図２参照）を例示している。機能合成ツール１２は、このアルゴリズム記述及び回路を構成する資源の制約条件を入力とする。この場合の資源制約条件は、下記の通りとする。

レジスタ：５個

加算器　：１個

図４に示されるアルゴリズム記述中に、変数はａ、ｂ、ｃ、ｄ、Ｘの５個、加算は３個が含まれる。資源制約条件から同時に使うことができる加算器は１つであるため、演算処理は時分割され、図５に示されるようになる。図５のＣｌｏｃｋ１ではａとｂの加算を行い、その結果をＸに代入する。Ｃｌｏｃｋ２では、Ｃｌｏｃｋ１と同じ加算器を用いて、ｃとｄの加算を行い、その加算結果を一時的に保持するために、変数ｔ２を新たに作成して代入する。更に、Ｃｌｏｃｋ３ではＣｌｏｃｋ１，２と同じ加算器を用いてＸ及びｔ２の加算を行い、その加算結果をＸに代入する。このような処理を演算器のス

ケジューリングと呼ぶ（ステップＳ２）。

【００３６】演算器のスケジューリングにより図４のアルゴリズム記述で示される処理を加算器１個の制約下で実現するためには、３クロックを要し、変数はａ，ｂ，ｃ，ｄ，Ｘ，ｔ２の６個を要することが分かる。レジスタに関する資源制約によれば、同時に利用できるレジスタは５個までとなっているため、レジスタについても共有化を行う必要がある。レジスタ共有化では、まずＣｌｏｃｋ１の変数ａ，ｂ及びＣｌｏｃｋ２の変数Ｘ，ｃ，ｄにレジスタ制約で許されるＲｅｇ１からＲｅｇ５までを割り当てる。Ｃｌｏｃｋ３の変数ｔ２については、Ｒｅｇ１からＲｅｇ５のうちのどれかを使い回す必要がある。Ｃｌｏｃｋ１で変数ａ，ｂに割り当てられていたＲｅｇ１、Ｒｅｇ２は、Ｃｌｏｃｋ２のＸが得られれば、その内容を保持する必要がないため、Ｃｌｏｃｋ２以降であれば使い回すことが可能である。ここでは、Ｃｌｏｃｋ３の変数ｔ２に割り当てるためＲｅｇ２を用いることにする。最後の演算結果変数Ｘについても同様にＣｌｏｃｋ３以降で使い回しができるＲｅｇ５を割り当てる。このようにして与えられたレジスタ制約の元でレジスタの共有を行う（ステップＳ３）。図４のアルゴリズム記述にステップＳ１、ステップＳ２、ステップＳ３の処理を行った結果得られるモデルをプログラミング言語で記述したものが下記に示されＦＳＭ／ＤａｔａＰａｔｈモデルとなる。

【００３７】図８は、変数／レジスタ／状態位置対応表２２を示している。変数／レジスタ／状態位置対応表２２は、図５に示される変数と、Ｃｌｏｃｋの対応、及び、図６に示されるレジスタとクロックの対応から作成される。図５及び図６のＣｌｏｃｋ１かＣｌｏｃｋ３は、図８の状態１～３に対応する。図８の状態１即ち図５、図６のＣｌｏｃｋ１では、レジスタＲｅｇ１は変数ａの値を持ち、レジスタＲｅｇ２は変数ｂの値を持つ。状態２では、レジスタＲｅｇ３は変数Ｘ、レジスタＲｅｇ４はｃ、レジスタＲｅｇ５はｄの値を持つ。状態３では、レジスタＲｅｇ２はｔ２、レジスタＲｅｇ３はＸ、状態４では、レジスタＲｅｇ５はＸの値を持つ。更に、初期状態として全てのレジスタに変数の割り当てがない状態０を作成する。このように各状態（即ちＣｌｏｃｋ）における変数とレジスタの対応を示したのが変数／レジスタ／状態位置対応表２２である（ステップＳ５）。

【００３８】図９は、ソース行／状態位置対応表２３を

```
int DataPath (int a,b,c,d, rest)
{
    static int FSM position;
    static int Reg1, Reg2, Reg3, Reg4, Reg5;
    if ( rest = = 1 ){
        FSM position = 0;
        return (0);
```

示している。Ｃｌｏｃｋ１における変数ａとｂの加算は、図４のアルゴリズム記述では３行目に当たる。Ｃｌｏｃｋ２における変数ｃとｄの加算は、図４のアルゴリズム記述では４行目に当たる。Ｃｌｏｃｋ３における変数Ｘとｔ２の加算も４行目に当たる。最終の演算結果Ｘが得られる状態は、図４のアルゴリズム記述の５行目に当たる。図８と同様に図９の状態１～３は、図５、図６のＣｌｏｃｋ１～３に対応する。このようにして得られるアルゴリズム記述中の行位置と状態の対応関係を示したのが、ソース行／状態位置対応表２３である。モデル変換ツール１３にモデルＩ／Ｆ情報２４、ＦＳＭ／ＤａｔａＰａｔｈモデル２１、変数／レジスタ／状態位置対応表２２、ソース行／状態位置対応表２３から、クロックレベルシミュレーションモデル８が作成される（ステップＳ７）。ここで、モデルＩ／Ｆ情報２４は、アルゴリズム記述３で表現される回路モジュールと外部とのインターフェースを定義する情報を持つ。その内容としてはバスからアクセス可能なコマンドレジスタや状態レジスタ、データメモリの構成、更には、他のモジュールとの間のデータ転送路のビット幅、同期方式などが含まれる。

【００３９】公知慣用のＲＴレベルシステム１６のＲＴＬ－ＨＤＬ１７は、図７に示されるように、機能合成ツール１２により自動生成される。公知ツールによりこのように自動生成されるハードウエア構成は、５つのレジスタ１～５と、７つのマルチプレクサ１～７と、５つのスイッチＳＷ１～５とから形成される。クロックレベル記述モデルよりも抽象度が低いこのようなハードウエアモデルでは、レジスタの値を保持するためにレジスタの出力をデータ入力にフィードバックさせるフィードバック機能とマルチプレクサとが必要であり、更に、全てのレジスタに常にクロック信号を供給するためのクロック制御と、各マルチプレクサを制御する制御信号４４とが必要であり、更に、演算器の共有化のために入力となる複数のレジスタから１つのレジスタを選択するためにもマルチプレクサが必要である。ハードウエアモデルは、後述されるように、その他の各種多様なハード部品とそれを制御する制御信号線が必要である。

【００４０】クロックレベルシミュレーションモデル８は、クロック単位で動作する言語として論理レベルの記述言語であるクロックレベル記述が下記ようになされる。

【００４１】

```
}
switch (FSM position) {
case 0:
        Reg1 = a; Reg2 = b; Reg3 = Reg1 + Reg2;
        FSM position = 1;
        break;
case 1:
        Reg4 = c; Reg5 = d; Reg2 = Reg4 + Reg5;
        FSM position = 2;
        break;
case 2:
        Reg5 = a; Reg3 + Reg2;
        FSM position = 3;
        break;
case3:
        X = Reg5;
        FSM position = 0;
        break;
```

【００４２】Ｃａｓｅ１はクロック１のステップに一致し、Ｃａｓｅ２はクロック２のステップに一致し、Ｃａｓｅ３はクロック３のステップに一致している。このようにクロック単位で、レジスタ１～５が分割されて使用されるクロックレベル記述が生成される。このクロックレベル記述は、図４に示されるアルゴリズム記述による表現よりも表現がより詳細になっているが、後記されるＲＴレベル記述から見れば、簡素化されている。そのアルゴリズムの内容によるが、アルゴリズム記述のシミュレーション時間はクロックレベル記述のシミュレーション時間に比べて概ね５００分の１であり、クロックレベル記述の動作時間はＲＴレベル記述の動作時間に比べて概ね５００分の１であることが本発明者により確認されている。

【００４３】図７に対応するＶＨＤＬ記述が、参考のために下記に記載される。ＲＴハードウエア記述１：

【数１】

```
entity FF is
  port(
    Reset:in   std_ulogic;
    Data :in   std_ulogic_vector(31 downto 0);
    Clock:in   std_ulogic;
    Q    :out  std_ulogic_vector(31 downto 0)
);
end FF;
```

【００４４】ＲＴハードウエア記述２：　　【数２】

```
architecture A of FF is
begin
  process(Reset, Clock)
  begin
    if(Reset='1')then
      Q <= '0';
    elsif(Clock'event)then
      if(Clock='1')then
        Q <= Data;
      elsif(Clock='X')then
        Q <= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
      endif;
    endif;
  end process;
end A;
```

【００４５】ＲＴハードウエア記述３：　【数３】

```
entity MUX is
  port(
    A   :in  std_ulogic_vector(31 downto 0);
    B   :in  std_ulogic_vector(31 downto 0);
    Sel :in  std_ulogic;
    Out :out std_ulogic_vector(31 downto 0)
  );
end MUX;

architecture A of MUX is
begin
  process(A, B, Sel)
  begin
    if(Sel='0')then
      Out <= A;
    elsif(Sel='1')then
      Out <= B;
    else
      Out <= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
    endif;
  end process;
end A;
```

【００４６】ＲＴハードウエア記述４：　【数４】

17 18

```
entity MUX3 is
  port(
    A   :in  std_ulogic_vector(31 downto 0);
    B   :in  std_ulogic_vector(31 downto 0);
    C   :in  std_ulogic_vector(31 downto 0);
    Sel :in  std_ulogic_vector(1 downto 0);
    Out :out std_ulogic_vector(31 downto 0)
  );
  end MUX3;
```

【００４７】ＲＴハードウエア記述５：　【数５】

```
architecture A of MUX3 is
begin
  process(A, B, C, Sel)
  begin
    if(Sel='00')then
      Out <= A;
    elsif(Sel='01')then
      Out <= B;
    elsif(sel='10')then
      Out <= C;
    else
      Out <= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
    endif;
  end process;
end A;
```

【００４８】ＲＴハードウエア記述６：　【数６】

```
entity ADDER is
  port(
    A   :in  std_ulogic_vector(31 downto 0);
    B   :in  std_ulogic_vector(31 downto 0);
    Out :out std_ulogic_vector(31 downto 0)
  );
  end ADDER;
```

【００４９】ＲＴハードウエア記述７：　【数７】

19 20

```
architecture A of ADDER is
begin
  Out <= A + B;
end A;

entity DATAPATH is
  port(
    a     :in  std_ulogic_vector(31 downto 0);
    b     :in  std_ulogic_vector(31 downto 0);
    c     :in  std_ulogic_vector(31 downto 0);
    d     :in  std_ulogic_vector(31 downto 0);
    Clock:in  std_ulogic;
    Reset:in  std_ulogic;
    X     :out std_ulogic_vector(31 downto 0)
);
end DATAPATH;
```

【００５０】ＲＴハードウエア記述８：　20　【数８】

```
architecture A of DATAPATH is
  component FF
    port(
      Reset :in  std_ulogic;
      Data  :in  std_ulogic_vector(31 downto 0);
      Clock :in  std_ulogic;
      Q     :out std_ulogic_vector(31 downto 0)
  );
  end component;
```

【００５１】ＲＴハードウエア記述９：　【数９】

```
  component MUX
    port(
      A   :in  std_ulogic_vector(31 downto 0);
      B   :in  std_ulogic_vector(31 downto 0);
      Sel :in  std_ulogic;
      Out :out std_ulogic_vector(31 downto 0)
  );
  end component;
```

【００５２】ＲＴハードウエア記述１０：　【数１０】

21 22

```
component MUX3
  port(
    A   :in  std_ulogic_vector(31 downto 0);
    B   :in  std_ulogic_vector(31 downto 0);
    C   :in  std_ulogic_vector(31 downto 0);
    Sel :in  std_ulogic_vector(1 downto 0);
    Out :out std_ulogic_vector(31 downto 0)
  );
  end component;
```

【００５３】ＲＴハードウエア記述１１：

【数１１】

```
component ADDER
  port(
    A   :in  std_ulogic_vector(31 downto 0);
    B   :in  std_ulogic_vector(31 downto 0);
    Out :out std_ulogic_vector(31 downto 0)
  );
  end component;
```

【００５４】ＲＴハードウエア記述１２：

【数１２】

```
component CONTOROLLER
  port(
    Clock:in  std_ulogic;
    Sel1 :out std_ulogic;
    Sel2 :out std_ulogic;
    Sel3 :out std_ulogic;
    Sel4 :out std_ulogic;
    Sel5 :out std_ulogic;
    Sel6 :out std_ulogic;
    Sel7 :out std_ulogic;
    Sel8 :out std_ulogic;
    Sel9 :out std_ulogic;
  );
  end component;
```

【００５５】ＲＴハードウエア記述１３：

【数１３】

23 24

```
--Register outputs
signal a_out       :std_ulogic_vector(31 downto 0);
signal b_out       :std_ulogic_vector(31 downto 0);
signal c_out       :std_ulogic_vector(31 downto 0);
signal d_out       :std_ulogic_vector(31 downto 0);
signal t1_out      :std_ulogic_vector(31 downto 0);
signal x_out       :std_ulogic_vector(31 downto 0);
--MUX select inputs
signal a_muxsel    :std_ulogic;
signal b_muxsel    :std_ulogic_vector(1 downto 0);
signal c_muxsel    :std_ulogic;
signal d_muxsel    :std_ulogic;
signal t1_muxsel   :std_ulogic;
signal add_in1_muxsel :std_ulogic_vector(1 downto 0);
signal add_in2_muxsel :std_ulogic_vector(1 downto 0);
signal x_muxsel    :std_ulogic;
--MUX outputs
signal a_muxout    :std_ulogic_vector(31 downto 0);
signal b_muxout    :std_ulogic_vector(31 downto 0);
signal c_muxout    :std_ulogic_vector(31 downto 0);
signal d_muxout    :std_ulogic_vector(31 downto 0);
signal t1_muxout   :std_ulogic_vector (31 downto 0);
signal add_in1_muxout :std_ulogic_vector(31 downto 0);
signal add_in2_muxout :std_ulogic_vector(31 downto 0);
signal x_muxout    :std_ulogic_vector(31 downto 0);
--operator in/out
signal add_in1     :std_ulogic_vector(31 downto 0);
signal add_in2     :std_ulogic_vector(31 downto 0);
signal add_out     :std_ulogic_vector(31 downto 0);
```

【００５６】ＲＴハードウエア記述１４： 【数１４】

```
begin
  X <= x_out;

  L_a_reg  :FF port map(Reset, a_muxout, Clock, a_out);
  L_b_t2_reg :FF port map(Reset, b_muxout, Clock, b_out);
  L_c_reg  :FF port map(Reset, c_muxout, Clock, c_out);
  L_d_reg  :FF port map(Reset, d_muxout, Clock, d_out);
  L_t1_reg :FF port map(Reset, t1_muxout, Clock, c_out);
  L_x_reg  :FF port map(Reset, x_muxout, Clock, x_out);

  L_a_mux  :MUX port map(a, a_out, a_muxsel, a_muxout);
  L_b_mux  :MUX3 port map(b, b_out, add_out, b_muxsel, b_muxout);
  L_c_mux  :MUX port map(c, c_out, c_muxsel, c_muxout);
  L_d_mux  :MUX port map(d, d_out, d_muxsel, d_muxout);
  L_t1_mux :MUX port map(add_out, t1_out, t1_muxsel, t1_muxout);
  L_x_mux  :MUX port map(x, x_out, x_muxsel, x_muxout);

  L_add_in1_mux :MUX3 port map(
          a_out, c_out, t1_out, add_in1_muxsel, add_in1);
  L_add in2_mux :MUX3 port map(
          b_out, d_out, t2_out, add_in2_muxsel, add_in2);

  L_adder  :ADDER port map(add_in1, add_in2, add_out);
end A;
```

【００５７】このような記述には、コントローラに関する記述は含まれていない。ＲＴハードウエア記述１４の最初の６行はＦＦレジスタに関する記述部分であり、その次の６行はマルチプレクサに関する記述部分であり、更にその次の記述はａｄｄｅｒによる加算に関する記述部分である。両レベルの記述の両分量は、概ね、両レベルのシミュレーションにかかる時間の長さに対応している。

【００５８】図１１は、アルゴリズムレベル記述ソース実行行表示ウインドウ４３にアルゴリズム記述ソースを表示する方法を示している。状態遷移が起こる毎に（ステップＳ１１）、ＧＵＩコントロール部４１は、現在の状態位置をＦＳＭ／ＤａｔａＰａｔｈ部３３に問い合わせる（ステップＳ１２）。例えば、現在の状態位置が状態２であった場合、ソース行／状態位置対応表２３から、対応するソースファイルがｆｉｌｅ１．ｃで行数が４行目であることが分かる（ステップＳ１３）。このようにして得られたソースファイルをＧＵＩ上で表示し、更に現在の状態位置に対応する行をハイライト表示することにより、ユーザーはアルゴリズムレベル記述上での現在の実行位置を示すことができる（ステップＳ１４）。

【００５９】図１２は、アルゴリズム記述中の変数をアルゴリズムレベル記述変数値表示ウインドウ４２に表示する方法を示している。状態遷移が起こる毎に（ステップＳ２１）、ＧＵＩコントロール部４１は、現在の状態位置をＦＳＭ／ＤａｔａＰａｔｈ部３３に問い合わせる（ステップＳ２２）。例えば、現在の状態位置が状態３であった場合、変数／レジスタ／状態位置対応表２２から、状態３で使われる変数はｔ２とＸであり、対応するレジスタがそれぞれＲｅｇ２とＲｅｇ３であることが分かる（ステップＳ２３）。ＧＵＩコントローラ部４１は、Ｒｅｇ２とＲｅｇ３の値をＦＳＭ／ＤａｔａＰａｔｈ部３３から得て、開けられている。アルゴリズム記述変数表示ウインドウ４２にはそれぞれｔ２及びＸの値として表示する（ステップＳ２４）。これにより、ユーザーはレジスタ共有が行われていても、ＧＵＩ上ではクロックレベルシミュレーションモデル中のレジスタ値の変化をアルゴリズムレベル記述上の変数の値の変化として観測することが可能になる。

【００６０】クロックレベルシミュレーションモデルは、図７の構造を持つＲＴＬ－ＨＤＬモデルに比べて、以下のシミュレーションが省略される。

（１）ＲＴＬ－ＨＤＬモデルは、レジスタ値を保持して

それを出力するためのフィードバックとマルチプレクサが必要であるが、プログラミング言語で実現されるクロックレベルシミュレーションモデルでは変数値の保持は代入が起こらない限りにおいて、フィードバックとマルチプレクサが不要である。

【００６１】（２）ＲＴＬ－ＨＤＬモデルは、全てのレジスタに常にクロック信号が供給されるためレジスタ値は新しいデータ入力値又はそのレジスタが保持していた値に常に更新されるようになるが、クロックレベルシミュレーションモデルでは新しいデータ入力があった変数のみが更新されるため、不要な更新処理がなくシミュレーションが高速化され得る。

【００６２】（３）ＲＴＬ－ＨＤＬモデルは、演算器の共有が行われるため、入力となる複数のレジスタから１つのレジスタを選択するマルチプレクサが必要であるが、クロックレベルシミュレーションモデルでは演算器に制限がなくそのような共有化が不要であり、演算器の入力を選択するマルチプレクサが不要である。

（４）クロックレベルシミュレーションモデルでは演算器共有化のためのマルチプレクサが不要であるので、マルチプレクサに必要であった制御入力を作成する回路が不要である。

【００６３】（５）ＲＴＬ－ＨＤＬモデルは、全レジスタに非同期リセット信号がつくが、クロックレベルシミュレーションモデルではクロック周期単位の動作のみを扱うので、非同期的な動作を行う必要がない。

【００６４】（６）ＲＴＬ－ＨＤＬモデルは、レジスタ，マルチプレクサ，演算器等のサブモジュールを利用した構造を持ち、そのサブモジュールは端子を持ち、その内部には信号線、制御機構等も持っているが、クロックレベルシミュレーションモデルでは、レジスタはプログラミング言語の変数に、マルチプレクサは条件文に、演算器は演算子で表現されるため、シミュレーション処理は大幅に簡略化される。

【００６５】（７）ＲＴＬ－ＨＤＬモデルは、実際のハードウエア通りに束線のビット位置の昇降順、符号のありなし、整数型／ビットベクタ型等の信号型の区別、又は、これらの間の変換を厳密に表現する必要があるが、このような区別は、クロックレベルシミュレーションモデルでは厳密に行われない。

【００６６】（８）ＲＴＬ－ＨＤＬモデル内にある各サブモジュール間の動作の並列性は厳密に表現されるが、クロックレベルシミュレーションモデルでは各サブモジュールの厳密な動作タイミングの差に基づいた検証は行われないため、並列性を厳密に表現する必要がない。

【００６７】（９）ＲＴＬ－ＨＤＬモデルは、クロックの変化タイミング以外にも例えばリセット信号が変化した時点の動作を厳密に表現するが、クロックレベルシミュレーションモデルではクロック単位の動作のみを扱うので、非同期的な動作に対する処理を簡略化することができる。

【００６８】

【発明の効果】本発明によるシミュレーションモデル、その記述とその生成の方法は、アルゴリズムレベルより抽象度が低くＲＴレベルよりも抽象度が高いレベルの言語の発見により、中間レベルのシミュレーションが可能になる。アルゴリズムレベルより抽象度が低くＲＴレベルよりも抽象度が高い記述に基づいて、アルゴリズム記述のシミュレーションよりもより精細に、且つ、ＲＴレベル記述のシミュレーションよりもより高速にシミュレートすることができる。一般的には、より物理的でありより詳細に記述されるハード部品のシミュレーション言語を論理化してハード部品を変数とするプログラミング言語を作成することが可能であり、公知の複数レベルの記述の中間言語を作成することができる。

【図面の簡単な説明】

【図１】図１は、本発明による論理シミュレーションシステムの実施の形態を示すシステム図である。

【図２】図２は、本発明によるシミュレーションモデル生成方法の実施の形態を示すブロックフロー図である。

【図３】図３は、２レベル間の移行のための対応を示す論理ブロック図である。

【図４】図４は、アルゴリズム記述の実施例を示す論理式である。

【図５】図５は、クロックレベルのスケジューリングをしめす動作時間表である。

【図６】図６は、クロックレベルのレジスタの動作を示す動作時間表である。

【図７】図７は、公知のハードウエア記述を示す回路図である。

【図８】図８は、状態遷移位置数とレジスタ変数値の対応を示すテーブルである。

【図９】図９は、状態遷移位置数とソース行との対応を示すテーブルである。

【図１０】図１０は、本発明による論理シミュレーション方法の動作の実施の形態を示すフローチャートである。

【図１１】図１１は、本発明による論理シミュレーション方法の他の動作を示すフローチャートである。

【図１２】図１２は、本発明による論理シミュレーション方法の更に他の動作を示すフローチャートである。

【図１３】図１３は、公知の論理シミュレーション方法を示すシステムフロー図である。

【符号の説明】

３…アルゴリズム記述

８…クロックレベル記述（クロックレベルシミュレーションモデル）

２２，２３…対応表

２２…変数／レジスタ／状態位置対応表

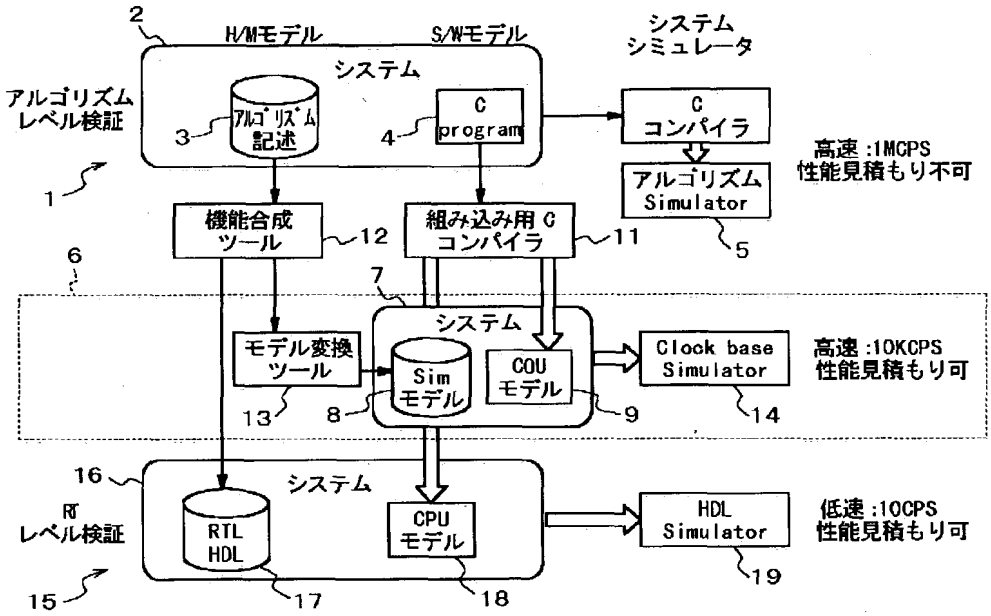
２３…ソース行／状態位置対応表

29　　30

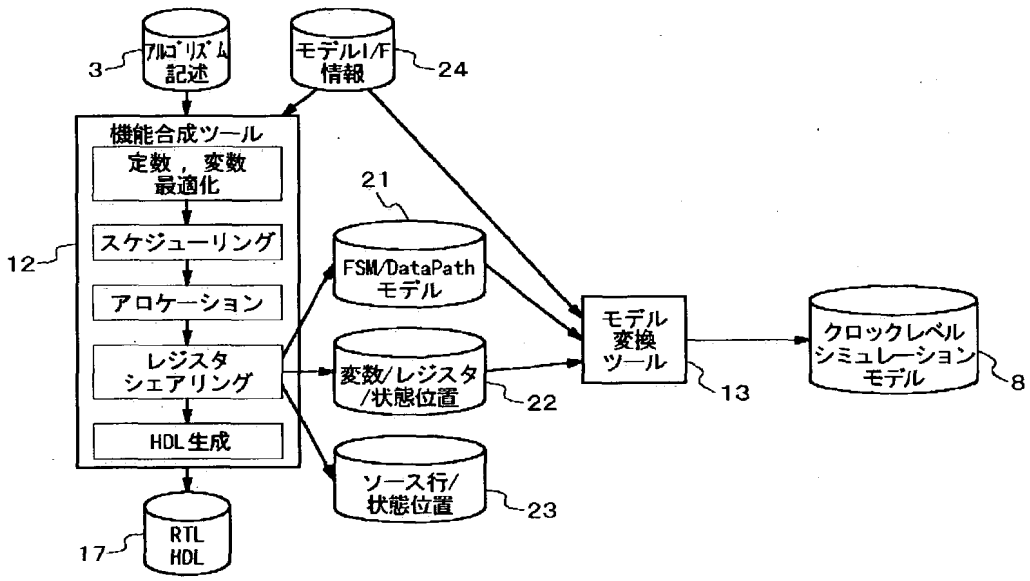
１７…ＲＴレベル記述　　Ｒｅｇ１～５…レジスタ

Ｒ…変数値

【図１】



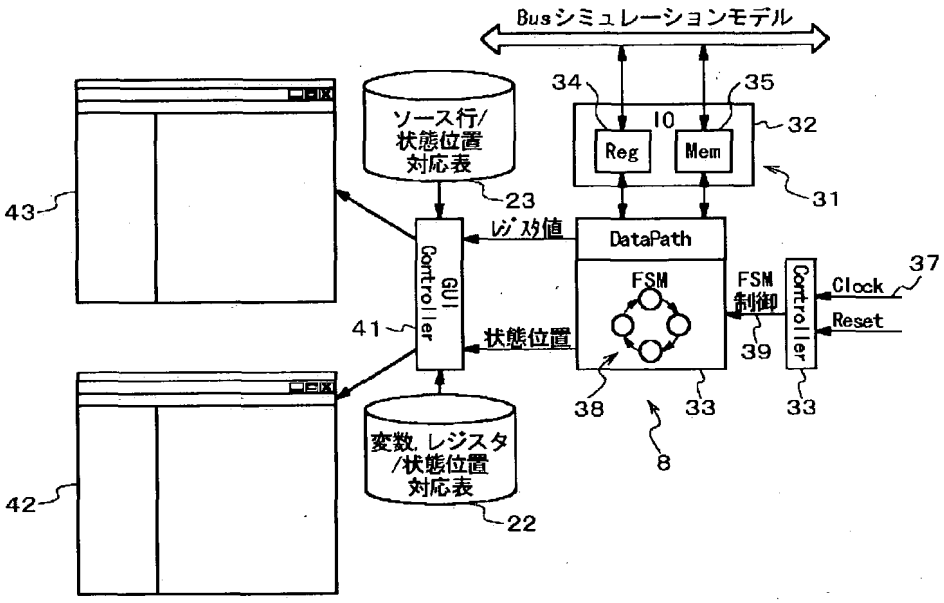
【図２】



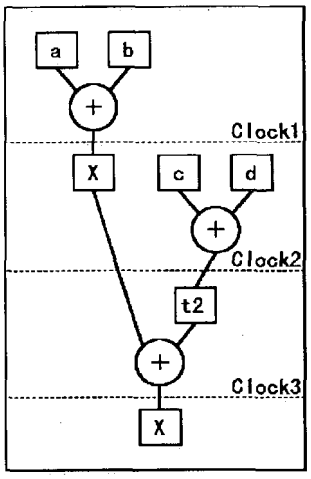
【図４】

```
1:int DataPath(int a,b,c,d)
2:{
3:      X=(a+b);
4:      X=(c+d)+X;
5:      return(X);
6:}
```

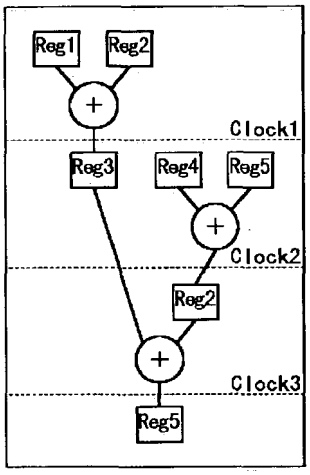
【図３】



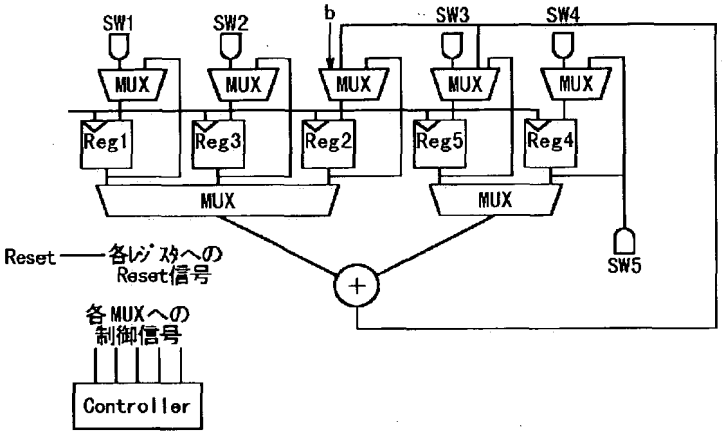
【図５】



【図６】



【図７】



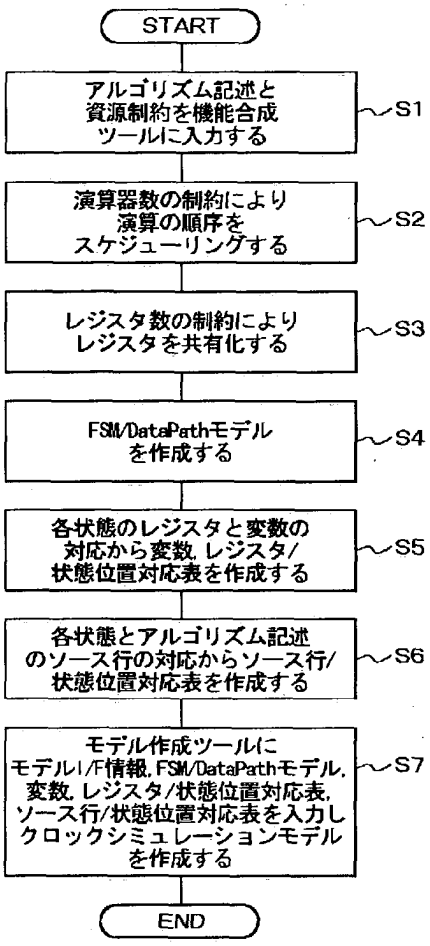
【図８】

	Reg1	Reg2	Reg3	Reg4	Reg5
状態 0	—	—	—	—	—
状態 1	a	b	—	—	—
状態 2	—	—	X	c	d
状態 3	—	t2	X	—	—
状態 4	—	—	—	—	X

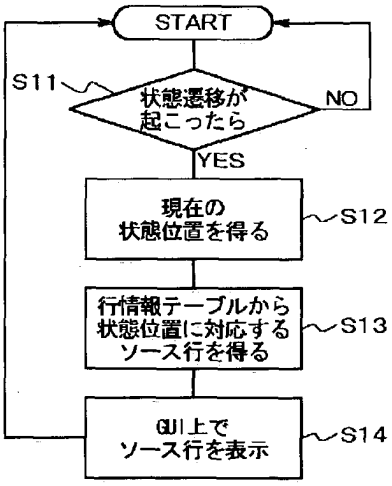
【図９】

	対応ソース行
状態 0	—
状態 1	File1.c:3
状態 2	File2.c:4
状態 3	File3.c:4
状態 4	File4.c:4

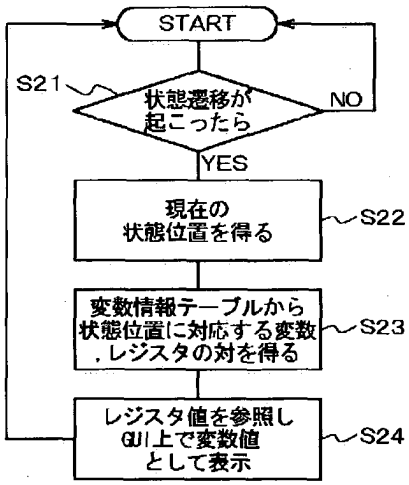
【図１０】



【図１１】



【図１２】



【図１３】